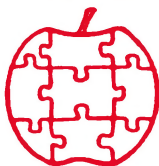


Apple

\$1.80



Assembly

Line

Volume 5 -- Issue 9

June, 1985

The Boyer-Morris String Search Algorithm	2
Short Integer Square Root Subroutine	13
Note on the TXS Instruction in the 65802	14
Interrupt Trace.	16
Improving the Single-Byte Converter.	21
AppleVisions, A Glimpse.	21
Two ROM Sets in One Apple //e.	22
Call Utility for Applesoft	24
Some Final DPl8 Subroutines.	28

S-C Macro Assembler ProDOS

We will begin shipping the ProDOS version of the S-C Macro Assembler in July, so we are now accepting advance orders. There is more to the ProDOS version than just a change of operating systems. The new upgrade includes a couple of major new features:

- .INB (INclude Blocked) directive -- This works just like .IN, except that only one disk block at a time is overlaid into memory. Allows assembly of much larger files, with only a minor speed penalty.
- .AC (Ascii Compressed) directive -- Generates compressed strings from a string between delimiters, according to rule tables. Very complex, but worth the effort if you have a lot of messages and need to save memory.

The price of the ProDOS version alone will be \$100. The upgrade from DOS Version 2.0 to ProDOS will be \$30. The upgrade from DOS Version 1.0 or 1.1 to ProDOS will be \$50, and will include DOS Version 2.0. The initial purchase price of the DOS 3.3 and ProDOS versions together will be \$120. These are introductory prices which may well be raised in a few months.

65802's Are Here!

After many months of manufacturing delays, Western Design Center is shipping 65802 and 65816 microprocessors. We recently received a final production '802, and it's now happily processing away in Bob's oldest Apple II (#219). You can order the chips from WDC for \$95.00. Call (602) 962-4545.

**The Boyer-Morris String Search Algorithm.....Bob Bernard
Westport, CT**

For years now, I have been working on a debugger for the Apple. Lately I have been adding a hex string search capability to it.

I needed one so I could look through the Apple IIc (ProDOS) utilities to see how it squirrels away in the alternate page screen holes user specified default settings for the serial ports. These are used at PR#1 or 2 time to simulate the dip switches on the Super Serial Card in a IIe. Without setting them you always get 9600 bps, etc. (Imagewriter settings, that is). I (and I assume other AAL readers) want a little routine for DOS 3.3 hello that will allow the user's defaults to be put away the same as the IIc utility does.

Well, that routine is not ready yet. However, the search utility is rather interesting in its own right.

I was just going to code up a straight hex search, but then I mentioned it to my computer science graduate son, David. He was horrified that I would waste my time on anything so crude. That's what I get for bringing up a programmer! David insisted that I should instead code an implementation of Boyer and Moore's algorithm, which appeared in the October 1977 issue of the Communications of the ACM. [A more recent reference is in the book "Algorithms", by Robert Sedgewick, (Addison-Wesley Publishing Co., 1983, 551 pages) on pages 249-252.]

Well, I read the article and it seemed like a challenge. Besides it looked like a real time saver, and could also be used for character string searches. The code here has been excerpted from my debugger, and then worked over by Bob S-C.

The "conventional" or "brute-force" search technique aligns the search pattern with the left end of the string to be searched through and compares one byte at a time, from left to right, until either the entire pattern is compared successfully or a mismatch occurs. In the latter case the search window is moved one byte to the right, and the comparing process is repeated.

Without any knowledge about the contents of the search pattern, the most the window can be moved is one place to the right. Boyer-Moore owes its speed advantage to the fact that it uses context (i.e. knowledge about the contents of the pattern to be searched for) to increase the distance that the search window can be advanced when a mismatch occurs. Thus efficiency increases as the length of the pattern increases, which does not happen in a conventional search.

The cost of this benefit (there always is a cost) is that a table (called DELTA in the CACM article and DELTA.TABLE in my program) is required to store this context information, 256 bytes in this implementation. One byte is needed in the table for every possible value of the characters in the string to be searched.

If a particular byte appears in the search pattern, then the

S-C Macro Assembler Version 2.0.....\$100
 Version 2.0 Upgrade Kit for 1.0/1 1/1 2 owners.....\$20
 Source Code for Version 1 1 (on two disk sides).....\$100
 Full Screen Editor for S-C Macro (with complete source code).....\$49
 S-C Cross Reference Utility (without source code).....\$20
 S-C Cross Reference Utility (with complete source code).....\$50
 DISASM Dis-Assembler (RAK-Ware).....\$30
 Source Code for DISASM.....additional \$30
 S-C Word Processor (with complete source code).....\$50
 DPl8 Source and Object.....\$50
 Double Precision Floating Point for Applesoft (with source code).....\$50
 S-C Documentor (complete commented source code of Applesoft ROMs).....\$50
 Source Code of //e CX & F8 ROMs on disk.....\$15

(All source code is formatted for S-C Macro Assembler. Other assemblers require some effort to convert file type and edit directives.)

AAL Quarterly Disks.....each \$15, or any four for \$45

		Jan-Mar	Apr-Jun	Jul-Sep	Oct-Dec
Each disk contains	1980	-	-	-	1
the source code from	1981	2	3	4	5
three issues of AAL,	1982	6	7	8	9
saving you lots of	1983	10	11	12	13
typing and testing.	1984	14	15	16	17
	1985	18	19		

AWiIe Toolkit (Don Lancaster, Synergetics).....\$39
 ES-CAPE: Extended S-C Applesoft Program Editor (new price, was \$60) \$40
 "Bag of Tricks", Worth & Lechner, with diskette.....(\$39.95) \$36
 MacASM -- Macro Assembler for Macintosh (Mainstay).....(\$150.00) \$100

Blank Diskettes (Verbatim)..... package of 20 for \$32
 (Premium quality, single-sided, double density, with hub rings)
 Vinyl disk pages, 6"x8.5", hold two disks each.....10 for \$6
 Diskette Mailing Protectors (hold 1 or 2 disks).....40 cents each
 (Cardboard folders designed to fit 6"x9" Envelopes.) or \$25 per 100
 Envelopes for Diskette Mailers..... 6 cents each

quikLoader EPROM System (SCRG).....(\$179) \$170
 PROMGRAMMER (SCRG).....(\$149.50) \$140
 D Manual Controller (SCRG).....(\$90) \$85
 Switch-a-Slot (SCRG).....(\$190) \$175
 Extend-a-Slot (SCRG).....(\$35) \$32

"Apple ProDOS: Advanced Features for programmers", Little..(\$17.95) \$17
 "Inside the Apple //c", Little.....(\$19.95) \$18
 "Inside the Apple //e", Little.....(\$19.95) \$18
 "Apple II+/Ile Troubleshooting & Repair Guide", Brenner(\$19.95) \$18
 "Apple II Circuit Description", Gayler.....(\$22.95) \$21
 "Understanding the Apple II", Sather.....(\$22.95) \$21
 "Understanding the Apple //e", Sather.....(\$24.95) \$23
 "Enhancing Your Apple II, vol. 1", Lancaster.....(\$15.95) \$15
 "Enhancing Your Apple II, vol. 2", Lancaster.....(\$17.95) \$17
 "Assembly Cookbook for the Apple II/Ile", Lancaster.....(\$21.95) \$20
 "Beneath Apple DOS", Worth & Lechner.....(\$19.95) \$18
 "Beneath Apple ProDOS", Worth & Lechner.....(\$19.95) \$18
 "6502 Assembly Language Programming", Leventhal.....(\$18.95) \$18
 "6502 Subroutines", Leventhal.....(\$18.95) \$18
 "Real Time Programming -- Neglected Topics", Foster.....(\$9.95) \$9
 "Microcomputer Graphics", Myers.....(\$12.95) \$12
 "Assem Language for Applesoft Programmers", Finley & Myers.(\$16.95) \$16
 "Assembly Lines -- the Book", Wagner(\$19.95) \$18
 "AppleVisions", Bishop & Grossberger.....(\$39.95) \$36

Add \$1 50 per book for US shipping. Foreign orders add postage needed.

Texas residents please add 6 1/8 % sales tax to all orders.

*** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
 *** (214) 324-2050 ***
 *** We accept Master Card, VISA and American Express ***

corresponding DELTA table entry contains the distance that the rightmost occurrence of that byte is from the left end of the pattern. All other entries contain the value -1. When a mismatch occurs, the DELTA table entry corresponding to that byte from the text being searched is used to compute how far to advance the search window. If that byte does not appear anywhere in the pattern, then the search window can be advanced by the full length of the pattern.

Since moving the search window, and the associated testing for finished, take most of the time in any searching technique, saving time here can be extremely beneficial, and explains why Boyer and Moore should be complimented.

My program uses the control-Y monitor command, in the form

```
adrl.adr2^Y <hexstring>
```

The two addresses specify the start and end of the area to be searched. "^Y" stands for "control-Y". The hex string may be separated from the control-Y by one or more spaces, if you desire. Since the control-Y doesn't show up on the screen, I usually type at least one space before the hex string. The hex string itself is a continuous string of hex digits, with no imbedded spaces. Here is an example that will search from \$800 to \$BFFF for "BERNARD":

```
800.BFFF^Y 4245524E415244
```

The program will list the starting addresses of any and all complete matches that are found.

The maximum length of the hex string is limited by the monitor input buffer. Since the longest command you can type is less than 256, and you have to use around ten characters for the addresses and control-Y, that puts an upper limit of less than 246 hex digits in your command. Each byte of the search pattern (or "key") is made up of two hex digits, so the maximum hex string will be less than 123 bytes long.

I assigned DELTA.TABLE to the area \$02D0.03CF. Since I scan and collect the search pattern right in the monitor keyboard buffer at \$0200, after converting to hex bytes it will run no higher than \$027F.

Actually, I only implemented a simplified version of Boyer and Moore's procedure. The CACM article also discusses a second table, DELTA2, which is filled with additional context information regarding "terminating substrings" of the search pattern. In cases where a partial mismatch occurs, it may be possible to advance the search window farther than the DELTA table would indicate. However, since such situations occur in less than 20% of the cases, David allowed that the potential additional speed did not justify the time and effort and the additional table and code space that would have been required, and he gave me a passing grade on my effort without it. The incorporation of this additional capability, and changes to make the program an ASCII search, are left "as a exercise for

the reader."

My program must go through several steps. First it has to find and pack up the search key. Next it must build the DELTA table. And finally the search can be performed.

Lines 1290-1360 will be executed when you BRUN the program. They install the control-Y vector and jump into the monitor, just as though you entered with CALL-151.

When you enter the search command, the Apple monitor parses the command line up to and including the control-Y, and then branches to my code at line 1380. The two addresses will have been converted and stuffed into A1 (\$3C,3D) and A2 (\$3E,3F). A variable named YSAV (at \$34) contains the index to the next character following the control-Y.

Lines 1400-1440 skip over any blanks you may have typed between the control-Y and the first hex digit. Actually, the Y-register gets incremented once too often, so lines 1460-1470 decrement Y and save it; now YSAV points to the first hex digit in the search key.

The next problem I had to solve was to differentiate odd from even length strings and arrange them properly, adding a leading zero when an odd number of hex digits is input. Lines 1490-1530 search for the end of the hex string; if there are no digits at all, we are finished and line 1530 returns for the next monitor command.

This is a nice place to insert a brief description of the NXTCHAR subroutine, found in lines 2460-2590. NXTCHAR picks up the next character from the input buffer, and tests to see if it is a hex digit. If so, it returns either \$00-09 or \$FA-FF in the A-register, and carry will be clear. If not a hex digit, it returns with carry set. If we got a digit, the Y-register indexing the input buffer will have been advanced.

Lines 1550-1590 compute the key length. Since two digits make a byte, the number of digits in the hex string divided by two gives the number of bytes. But I actually want to use the byte-count-minus-one. Also I need to adjust for odd or even length strings. Lines 1600-1650 take care of these details. If the count was odd, I jump into the middle of the packing loop so that a leading zero gets inserted.

Lines 1670-1800 comprise the packing loop. NXTCHAR will return with carry set when we try to get a digit beyond the end of the key, so line 1680 is the only test in the loop. Lines 1670-1730 retrieve a left-hand digit and store it in the buffer. Lines 1740-1800 do the same for right-hand digits. Key bytes are stored starting at \$0200, so they never catch up to the advancing retrieval of digits.

Line 1810 sets YSAV to point to the first character past the end of the hex string. This will usually be a carriage return, or another monitor command. Unless it is beyond \$2CF, the monitor will correctly continue parsing whatever is in the

buffer when we are through searching. At \$2D0 and beyond, the DELTA table will clobber any further characters.

Now we come to the Boyer-Moore part. Lines 1820-1870 initialize the DELTA table to all -1 values, which is what we want for any bytes not present in the key. When the loop finishes, X=0 again.

Lines 1880-1970 scan through the search key from left to right, and store into DELTA the index of the rightmost occurrence of each value in the key. For example, if the key is "4245524E415244" ("BERNARD" again), the DELTA values will be:

```
DELTA+$41: 4
DELTA+$42: 0
DELTA+$44: 6
DELTA+$45: 1
DELTA+$4E: 3
DELTA+$52: 5 (also at 2, but 5 is rightmost)
all others: -1
```

We'll continue with this example after a brief look at the rest of the code.

Lines 1980-2040 back up the end pointer, which has been patiently waiting all this time in A2L and A2H. We subtract the key length (in bytes, not digits) from the end pointer, so that we will not try to match the key any further than necessary. We could do this inside the search loop, but it will run faster if we do it once before the loop.

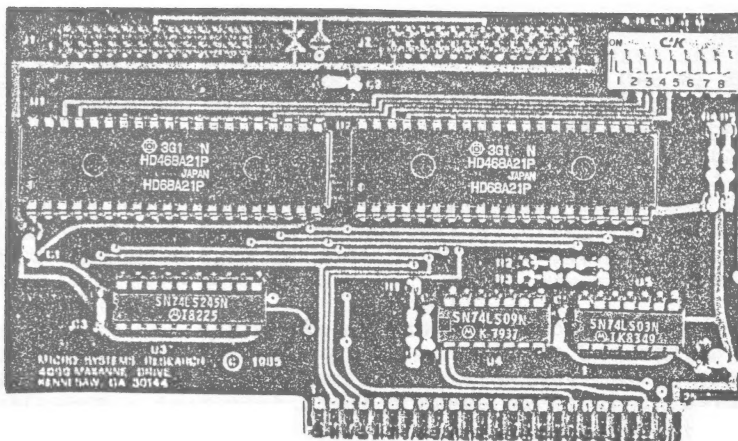
Lines 2050-2440 perform the search. I inserted lines 2070-2110 inside the loop to printout the search window start address each time through the loop. This helps me to make sure it is working, and to explain how. Of course you should remove these five lines before using the routine for real problems. Notice they are all marked "<<<DEBUG>>>".

Lines 2120-2170 check whether the beginning of the search window has moved past the end of the area to be searched. If so, we are finished.

Lines 2180-2240 compare bytes from the key and the search window. If the entire key matches, we fall out of the loop into lines 2250-2300, where the address of the match will be printed. After a successful match the search window will be moved one byte to the right by lines 2370-2430, and we will begin the SEARCH.LOOP again.

Notice that the key is compared from right-to-left, not left-to-right. This is a critical part of the Boyer-Moore method. If a key byte does not match a search-window byte, we branch to line 2320. The byte from the search window is in the A-register. Lines 2320-2370 compute how far we can advance the search window, based on just what character we DID find in the search window, and how far into the key we had already matched.

To see how this works, let's continue the "BERNARD" example.



*** MICROPORT (TM) 32 ***

---NEW--- APPLE COMPATIBLE DIGITAL I/O CARD ---NEW---

- * PROVIDES 32 I/O LINES / FOUR 8-BIT PORTS TO 2 CONNECTORS.
- * EACH PORT HAS 2 HANDSHAKE LINES FOR ERROR FREE COMMUNICATION WITH EXTERNAL DEVICES.
- * INTERRUPT ARBITRATION CIRCUIT TO ASSURE RELIABILITY OF INTERRUPTING SOURCE.
- * ALL INPUTS AND OUTPUTS ARE TTL COMPATIBLE.
- * COMES WITH FULL DOCUMENTATION AND SAMPLE PROGRAMS IN BASIC AND 6502 MACHINE LANGUAGE.
- * PROVIDES +5 VOLTS ON CONNECTORS FOR EXTERNAL DEVICES OR RELAYS.
- * THIS BOARD IS AN EXCELLENT CHOICE FOR ANY DIGITAL MONITORING AND/OR CONTROL APPLICATION INCLUDING BURGLAR ALARMS, PRINTERS, PLOTTERS, DIGITAL JOYSTICK, TO CONTROL HEAVY LOADS ie. MOTORS, AC LIGHTS, USE WITH RELAYS.
- * HIGH QUALITY GLASS EPOXY PC BOARDS, SOLDERMASK AND GOLD CONTACTS
- * INCLUDES THREE YEAR WARRANTY ON ASSEMBLED PRODUCTS.

VISA-MASTER CARD-C.O.D.-ACCEPTED
GA RESIDENTS ADD 3% TAX
OR SEND CHECK/MONEY ORDER TO:

ASSEMBLED AND TESTED: \$88.00
COMPLETE KIT: \$65.00

MICRO SYSTEMS RESEARCH
4099 MAXANNE DR.
KENNESAW GA 30144

FOR ADDITIONAL INFORMATION OR
TO PLACE AN ORDER CALL:
(404) 928-9394

Suppose the text we are searching is "THERE ARE FEW ST. BERNARDS IN SAN BERNARDINO." The key will be BERNARD, entered in hex as shown above. We first try to match BERNARD at the beginning of the text. We start at the right end, matching the "D" of the key with "A" of the text. The match fails, so we look up the "A" value in the DELTA table, which is 4. We subtract the delta value (4) from the current key index (6) and add the result (6-4=2) to the search window address. Note that this has the result of aligning the "A" of BERNARD with the "A" in the text.

Back to the top, and we now try to match the "D" of BERNARD to the "E" at the end of "ARE". Failure again! This time the DELTA value is 1, and we are still at position 6 in the key: index-delta is 5, so we advance the window by 5. This lines up the "E" of BERNARD with the E of the text. The next attempted match will find a blank in the text, which does not occur in the key at all. The delta value for blank is -1: $6 - (-1) = 7$. so we will advance the window by 7. Now the window is up to "ST. BER" in the text.

When we compare "D" of BERNARD to "R" in the text, we fail again. The delta value for R is 5. There are two R's in BERNARD, but the rightmost one is at index 5. We can move the search window by $6 - 5 = 1$. Next we try "D" against "N". The delta value of "N" is 3, so we can move the window $6 - 3 = 3$ bytes. This time we have found "BERNARD"!

If you count it all up, we have compared the "D" of BERNARD with only six characters, and already we are at the first occurrence of the whole key in the text. A conventional search would have tried to match the first character of the key ("B") with all 18 characters in the text which precede the first "B" of the text. We have saved 13 times around the main loop! Of course, our loop is a tiny bit longer, but the end result is faster.

Here is a step-by-step picture of the entire search, which finds BERNARD twice:

```

THERE ARE FEW ST. BERNARDS IN SAN BERNARDINO.
  BERNARD
    BERNARD
      BERNARD
        BERNARD
          BERNARD (success!)
            BERNARD
              BERNARD
                BERNARD
                  BERNARD
                    BERNARD (success!)
                      BERNARD
                        BER... (end)

```

I have tacked two more examples onto the end of the source code, at lines 2620-2690. You can play with them. The five <<<DEBUG>>> lines will print out the window address at each step, so you can see how the search progresses. Remember to

take those lines out before you make a production version of the program.

If you decide to include this search algorithm in your own private debugger program, like I am, you might want to add the ability to use an ASCII string for the key. You could use a quotation mark after the control-Y to signal the packer loop that an ASCII string follows. You might also want to add single-byte wildcard characters, and/or the ability to ignore the high-order bit of each byte matched.

Perhaps the Boyer-Moore algorithm would be even more useful in a data base program, a word processor, or other context in which you are searching through huge quantities of text for relatively interesting keys. My example should get you started, and my son will be proud of you!

```

1000 *SAVE S.HEX.SEARCH
1010 *-----
1020 *   MEMORY SEARCH FOR HEX STRING
1030 *       BY BOB BERNARD, MAY 17, 1985
1040 *       MODIFIED BY BOB S-C, MAY 27TH
1050 *   ADR1-ADR2 YXXXXXXXXXX
1060 *       ("Y" MEANS CONTROL-Y)
1070 *
1080 *   SEARCH MEMORY FROM ADR1 THRU ADR2
1090 *   LOOKING FOR REFERENCES TO
1100 *   THE HEX STRING, YXXXXXXXXX
1110 *
1120 *-----
34- 1130 YSAV          .EQ $34
3C- 1140 A1L          .EQ $3C,3D   START OF SEARCH AREA
3E- 1150 A2L          .EQ $3E,3F   END OF SEARCH AREA
40- 1160 KEY.LENGTH   .EQ $40      (MONITOR'S A3L)
1170 *-----
0200- 1180 KBDBUF      .EQ $0200 THRU $2CF
02D0- 1190 DELTA.TABLE .EQ $02D0 THRU $3CF
03F8- 1200 USRADR      .EQ $03F8   CTL-Y JUMPS HERE
1210 *-----
F941- 1220 PRINTAX     .EQ $F941
FD8E- 1230 CROUT       .EQ $FD8E   NEW LINE
FF69- 1240 MONZ        .EQ $FF69   MONITOR, NO BELL
1250 *-----
1260         .OR $0800
1270         .TF B.HEX.SEARCH
1280 *-----
1290 HEX.SEARCH
1300 LDA #$4C          JMP OPCODE
0802- 08D F8 03 1310 STA USRADR      STUFF INTO CNTL-Y EXIT LOC
0805- 089 12      1320 LDA #SEARCH     LO ADR
0807- 08D F9 03 1330 STA USRADR+1
080A- 089 08      1340 LDA /SEARCH    HI ADR
080C- 08D FA 03 1350 STA USRADR+2
080F- 08C 69 FF 1360 JMP MONZ        MONITOR, NO BELL
1370 *-----
1380 SEARCH
1390 *---SKIP LEADING BLANKS-----
0812- A4 34      1400 LDY YSAV          NEXT VALID KBDBUF CHAR
0814- B9 00 02 1410 .1 LDA KBDBUF,Y    GET CHAR FROM
0817- C8          1420 INY            KEYBOARD BUFFER
0818- C9 A0      1430 CMP # " "      SKIP LEADING BLANKS
081A- F0 F8      1440 BEQ .1
1450 *---MARK KEY START-----
081C- 88          1460 DEY
081D- 84 34      1470 STY YSAV          WHERE SCAN STARTS
1480 *---FIND END OF KEY-----
081F- 20 C2 08 1490 .2 JSR NITCHAR
0822- 90 FB      1500 BCC .2             ...HEX DIGIT
0824- C4 34      1510 CPY YSAV          CHECK FOR NULL KEY
0826- D0 01      1520 BNE .3             ...NOT NULL
0828- 60          1530 RTS            NULL KEY

```

```

1540 *---COMPUTE KEY LENGTH-----
0829- 98 1550 .3 TYA
082A- E5 34 1560 SBC YSAV
1570
082C- 4A 1580 LSR
082D- 85 40 1590 STA KEY.LENGTH
082F- A4 34 1600 LDY YSAV
0831- A2 00 1610 LDX #0
0833- 8E 00 02 1620 STX KBDBUF (IN CASE ODD COUNT)
0836- B0 0E 1630 BCS .5 ...ODD NUMBER OF BYTES
1640 *---ADJUST FOR EVEN LENGTH-----
0838- C6 40 1650 DEC KEY.LENGTH MAKE EVEN LENGTH ONE LESS
1660 *---LEFT NYBBLE-----
083A- 20 C2 08 1670 .4 JSR NITCHAR
083D- B0 15 1680 BCS .6 END OF KEY
083F- 0A 1690 ASL
0840- 0A 1700 ASL
0841- 0A 1710 ASL
0842- 0A 1720 ASL
0843- 9D 00 02 1730 STA KBDBUF,X LEFT HALF DEST CHAR
1740 *---RIGHT NYBBLE-----
0846- 20 C2 08 1750 .5 JSR NITCHAR
0849- 29 0F 1760 AND #$0F
084B- 1D 00 02 1770 ORA KBDBUF,X MERGE HI NIBBLE
084E- 9D 00 02 1780 STA KBDBUF,X
0851- E8 1790 INX
0852- D0 E6 1800 BNE .4 ...ALWAYS
0854- 84 34 1810 .6 STY YSAV
1820 *---INIT ALL DELTAS--1-----
0856- A2 00 1830 LDX #0
0858- A9 FF 1840 LDA #-1
085A- 9D D0 02 1850 .7 STA DELTA.TABLE,X
085D- E8 1860 INX
085E- D0 FA 1870 BNE .7 ...256 OF THEM
1880 *---DELTA(KEY(I))=I-----
0860- A0 00 1890 LDY #0 FOR I=0 TO KEYLEN
0862- B9 00 02 1900 .8 LDA KBDBUF,Y DELTA(K) = DISTANCE FROM LEFT END
0865- AA 1910 TAX OF RIGHT-MOST OCCURENCE OF
0866- 98 1920 TYA 8-BIT VALUE "K" IN KEY.
0867- 9D D0 02 1930 STA DELTA.TABLE,X
086A- C8 1940 INY NEXT I
086B- C4 40 1950 CPY KEY.LENGTH
086D- 90 F3 1960 BCC .8
086F- F0 F1 1970 BEQ .8
1980 *---ADJUST END OF SEARCH-----
0871- 38 1990 SEC
0872- A5 3E 2000 LDA A2L
0874- E5 40 2010 SBC KEY.LENGTH
0876- 85 3E 2020 STA A2L
0878- B0 02 2030 BCS SEARCH-LOOP
087A- C6 3F 2040 DEC A2L+1
2050 *-----
2060 SEARCH-LOOP
087C- A5 3D 2070 LDA A1L+1 <<<DEBUG>>>
087E- A6 3C 2080 LDX A1L <<<DEBUG>>>
0880- 20 41 F9 2090 JSR PRINTAX <<<DEBUG>>>
0883- A9 AD 2100 LDA #"-> <<<DEBUG>>>
0885- 20 ED FD 2110 JSR $FDED <<<DEBUG>>>
0888- A5 3E 2120 LDA A2L CHECK AGAINST
088A- C5 3C 2130 CMP A1L UPPER BOUND
088C- A5 3F 2140 LDA A2L+1 FOR SEARCH
088E- E5 3D 2150 SBC A1L+1
0890- B0 01 2160 BCS .1 A1<=A2, NOT FINISHED
0892- 60 2170 RTS A1>A2, FINISHED
2180 *---COMPARE IN THIS POSITION-----
0893- A4 40 2190 .1 LDY KEY.LENGTH FOR I=KEYLEN TO 0
0895- B1 3C 2200 .2 LDA (A1L),Y CHECK BYTES FROM
0897- D9 00 02 2210 CMP KBDBUF,Y RIGHT TO LEFT
089A- D0 10 2220 BNE .3 ...DID NOT MATCH
089C- 88 2230 DEY NEXT I
089D- 10 F6 2240 BPL .2
2250 *---MATCH FOUND-----
089F- A6 3C 2260 LDX A1L PRINT ADR
08A1- A5 3D 2270 LDA A1L+1 WHERE MATCH
08A3- 20 41 F9 2280 JSR PRINTAX WAS FOUND
08A6- 20 8E FD 2290 JSR CROUT NEW LINE
08A9- 4C B4 08 2300 JMP .4

```



DISASM 2.2e - AN INTELLIGENT DISASSEMBLER : \$30.00

Investigate the inner workings of machine language programs. DISASM converts machine code into meaningful, symbolic source. Creates a standard text file compatible with S-C, LISA, ToolKit and other assemblers. Handles data tables, displaced object code & even lets you substitute your own meaningful labels. (100 commonly used Monitor and Pg Zero names included.) An address-based triple cross reference table is provided to screen or printer. DISASM is an invaluable machine language learning aid to both novice & expert alike. Don Lancaster says DISASM is "absolutely essential" in his new **ASSEMBLY COOKBOOK**. For entire Apple II family including the new Apple IIc (with all the new opcodes). **SOURCE CODE** available for an additional \$30.00

LOW LOW PRICE !!! C-PRINT For The APPLE IIc : \$69.00

Connect standard parallel printers to an Apple IIc. C-PRINT is a hardware accessory that plugs into the standard Apple IIc printer serial port. The other end plugs into any printer having a standard 36 pin centronics-type parallel connector. Just plug in and print! High speed data transfer at 9600 Baud. No need to reconfigure serial port or load software drivers for text printing.

FONT DOWNLOADER & EDITOR : \$39.00

Turn your printer into a custom typesetter. Downloaded characters remain active while printer is powered. Use with any Word Processor program capable of sending ESC and control codes to printer. Switch back and forth easily between standard and custom fonts. All special printer functions (like expanded, compressed etc.) apply to custom fonts. Full HIRES screen editor lets you create your own characters and special graphics symbols. Compatible with many parallel printer I/F cards. User driver option provided. For Apple II, II+, IIe. Specify printer: Apple Dot Matrix, C.Itoh 8510A (Prowriter), Epson FX 80/100, or OkiData 92/93.

The Font Downloader & Editor for the Apple Imagewriter Printer. For use with Apple II, II+, IIe (with SuperSerial card) and the new Apple IIc (with builtin serial interface).

FONT LIBRARY DISKETTE #1 : \$19.00 Contains lots of user-contributed fonts for all printers supported by the Font Downloader & Editor. Specify printer with order.

The 'PERFORMER' CARD : \$39.00

Plugs into any slot to convert a 'dumb' centronics-type printer I/F card into a 'smart' one. Command menu eliminates need to remember complicated ESC codes. Features include perforation skip, auto page numbering with date & title. Includes large HIRES graphics & text screen dumps. Specify printer: MX-80 with Grafbax-80, MX-100, MX-80/100 with Grafbaxplus, NEC 8092A, C.Itoh 8510 (Prowriter), OkiData 82A/83A with Okigraph & OkiData 92/93. **SOURCE CODE : \$30.00**

FIRMWARE FOR APPLE-CAT: The 'MIRROR' ROM : \$25.00

Communications ROM plugs directly into Novation's Apple-Cat Modem card. Basic modes: Dumb Terminal, Remote Console & Programmable Modem. Features include: selectable pulse or tone dialing, true dialtone detection, audible ring detect, ring-back, printer buffer, 80 col card & shift key mod support. Uses superset of Apple's Comm card and Micromodem II commands. **SOURCE CODE : \$50.00**

RAM/ROM DEVELOPMENT BOARD : \$30.00

Plugs into any Apple slot. Holds one user-supplied 2Kx8 memory chip (6116 type RAM for program development or 2716 EPROM to keep your favorite routines on-line). Maps into \$Cn00-CnFF and \$C800-CFFF.

ALL NEW !!! MIDI MUSIC PRODUCTS

MIDI means Musical Instrument Digital Interface. Use your computer with any MIDI-equipped music keyboard for entertainment and music education. Low cost MIDI player interface cable, complete with 6 song demo disk: \$49.00. Thousands of popular songs available soon on diskette (also compatible with Passport MIDI interface). Products for both the Apple IIc and Commodore 64/128. Unique general purpose MIDI expander cable and gender changer also available. Send SASE for product descriptions and prices.

Avoid a \$3.00 handling charge by enclosing full payment with order. VISA/MC and COO phone orders accepted.

RAK-WARE 41 Ralph Road W. Orange N.J. 07052 (201) 325-1885



```

2310 #---ADVANCE SEARCH POINTER-----
08AC- AA 2320 .3 TAX STRING CHAR JUST LOOKED AT
08AD- 98 2330 TYA I
08AE- 18 2340 CLC
08AF- FD DO 02 2350 SBC DELTA.TABLE.X
08B2- 10 02 2360 BPL .5 ...VALUE IS POSITIVE
08B4- A9 00 2370 LDA #0 ...ADVANCE BY 1
08B6- 38 2380 .5 SEC COMPENSATE
08B7- 65 3C 2390 ADC A1L
08B9- 85 3C 2400 STA A1L
08BB- 90 BF 2410 BCC SEARCH.LOOP
08BD- E6 3D 2420 INC A1L+1
08BF- D0 BB 2430 BNE SEARCH.LOOP ...ALWAYS, UNLESS WE
08C1- 60 2440 RTS ...RAN OFF THE END OF MEMORY
2450 #-----
2460 N1TCHAR
08C2- B9 00 02 2470 LDA KBDBUF,Y NEXT ACTIVE CHAR
08C5- C8 2480 INY
08C6- 49 B0 2490 EOR #$B0 CONVERT ASCII TO DIGIT
08C8- C9 0A 2500 CMP #10 0..9?
08CA- 90 09 2510 BCC .1 YES
08CC- 69 88 2520 ADC #$88 SHIFT RANGE FOR A-F TEST
08CE- C9 FA 2530 CMP #$FA A..F?
08D0- B0 03 2540 BCS .1 YES. EXIT CC
08D2- 38 2550 SEC NOT HEX CHAR
08D3- 88 2560 DEY BACK UP INDEX
08D4- 60 2570 RTS
08D5- 18 2580 .1 CLC SIGNAL HEX CHAR
08D6- 60 2590 RTS
2600 #-----
08D7- 2610 END .BS $A00-*
2620 TEST.STRING
0A00- 58 58 58
0A03- 58 58 58
0A06- 58 43 4F
0A09- 43 41 43
0A0C- 41 43 41
0A0F- 43 41 43
0A12- 41 43 41
0A15- 43 41 43
0A18- 41 43 41
0A1B- 43 41 43
0A1E- 41 43
2630 .AS /XXXXXXXXCOCACACACACACACACACAC/
2640 * TRY A00.A1F`Y 43414341434143
2650 * SHOULD GET A09-A0B-A0D-A0F-A11-A13-A15-A17-A19
2660 #-----
0A20- 41 20 53
0A23- 54 52 49
0A26- 4E 47 20
0A29- 53 45 41
0A2C- 52 43 48
0A2F- 49 4E 47
0A32- 20 45 58
0A35- 41 4D 50
0A38- 4C 45 20
0A3B- 43 4F 4E
0A3E- 53 49 53
0A41- 54 49 4E
0A44- 47 20 4F
0A47- 46 20 53
0A4A- 49 4D 50
0A4D- 4C 45 20
0A50- 54 45 58
0A53- 54
2670 TS2 .AS /A STRING SEARCHING EXAMPLE CONSISTING OF SIMPLE TEXT/
2680 * TRY A20.A53`Y48494E47
2690 #-----

```

Note About Alliance Computers

Back in January Alliance Computers advertised 65802's for \$50.00, but couldn't fill the orders because the chips didn't exist yet. Some of their formerly unhappy customers tell me that their orders have now arrived, so Alliance is taking care of their customers. You can reach Alliance Computers at P.O. Box 408, Corona. NY 11368.

Short Integer Square Root Subroutine.....Bob Sander-Cederlof

In some graphics situations you need a square root subroutine (it is probably the fault of Pythagoras). Since the screen coordinates are integers, a short and fast integer square root subroutine can be handy.

The following program is probably not in the "fast" category, but it is indeed short. It can produce the integer value of the square root of any integer from 0 through 65535. The program uses the method of subtracting successive odd numbers.

Every perfect square (N^2 , where N is an integer) is the sum of a series of odd numbers from 1 through $2*N-1$. Thus $4=1+3$, $25=1+3+5+7+9$, etc.

The program starts by subtracting 1, then 3, then 5, and so on until the remainder is negative. When the remainder goes negative, the last odd number subtracted was $2*N+1$, so we can get the square root by dividing that odd number by 2.

I set up the routine so I could test it with an Applesoft program. You can POKE the low 8-bits of a number at 768 (\$300), the high 8-bits at 769, and CALL 772. Upon return, PEEK(770)+256*PEEK(771) gives you the integer value of the square root.

I used a couple of tricks in the code. For one, the variable ODD is always an even number. Since I preface the subtraction with CLC, a "borrow" is assumed, so it has the effect of subtracting the odd number which is one larger than the even number in ODD. This save a LDA #1 instruction after line 1090.

In lines 1190-1230, I add 2 to the even number in ODD. But you can see that line 1200 is ADC #1. This adds 2 because carry happens to be set.

```

1000 *SAVE S.SQRT16
1010 *-----
1020                .OR $300
0300- 1030 ARG      .BS 2
0302- 1040 ODD      .BS 2
1050 *-----
0304- AE 01 03 1060 SQRT  LDX ARG+1  X = HI BYTE  HI
0307- AC 00 03 1070 LDY ARG      Y = LO BYTE  LO
030A- A9 00 1080 LDA #0        START ODD=0
030C- 8D 03 03 1090 STA ODD+1
030F- 8D 02 03 1100 .1 STA ODD
0312- 18 1110 CLC              BORROW ON. SUBTRACT (ODD+1)
0313- 98 1120 TYA              LO
0314- ED 02 03 1130 SBC ODD
0317- A8 1140 TAY
0318- 8A 1150 TXA              HI
0319- ED 03 03 1160 SBC ODD+1
031C- AA 1170 TAX
031D- 90 0C 1180 BCC .2        ...ODD>REMAINDER. FINISHED
031F- AD 02 03 1190 LDA ODD      CARRY SET, ADD 2 TO ODD
0322- 69 01 1200 ADC #1
0324- 90 E9 1210 BCC .1        ...NEXT
0326- EE 03 03 1220 INC ODD+1
0329- D0 E4 1230 BNE .1        ...ALWAYS
032B- 4E 03 03 1240 .2 LSR ODD+1 SQRT IS (ODD/2)
032E- 6E 02 03 1250 ROR ODD
0331- 60 1260 RTS
1270 *-----

```

Note on the TXS instruction in the 65802...Bob Sander-Cederlof

Sandy Greenfarb wrote the other day that he had received a 65802 and plugged it into his Basis 108 with success.

He has been trying various permutations of the new opcodes and modes, and discovered some stones are better left unturned:

"The following programs should both print the letter "A" on the screen. However, the one on the left works, while the one on the right hangs up the computer."

Works -----	Hangs Up -----
CLC	CLC
XCE	XCE
LDA #*A	LDA #*A
JSR \$FDF0	JSR \$FDED
SEC	SEC
XCE	XCE
RTS	RTS

The only difference in the two programs is that the unsuccessful one weaves its way through DOS. I looked at the DOS code it goes through, and at first glance it appears there should be NO PROBLEMS associated with executing all this code in 65802 mode, since both 16-bit modes are off.

However, for some reason it still hangs up. Actually, it might not always hang: it depends on what is in page zero at the corresponding position as the stack pointer in page one.

Complete, Commented Source Code!

Our software is not only unlocked and fully copyable

...we often provide the complete source code on disk, at unbelievable prices!

S-C Macro Assembler. The key to unlocking all the mysteries of machine language. Combined editor/assembler with 29 commands, 20 directives. Macros, conditional assembly, global replace, edit, and more. Highest rating "The Book of Apple Software" in 1983 and 1984. \$80.

Powerful cross-assembler modules also available to owners of S-C Macro Assembler. You can develop software on your Apple for 6800, 6805, 6809, 68000, 8085, 8048, 8051, 1802, LSI-11, and Z-80 microprocessors. \$50 each.

S-C Xref. A support program which works with the S-C Macro Assembler to generate an alphabetized listing of all labels in a source file, showing with each label the line number where it is defined along with all line numbers containing references to the label. You get the complete source code for this amazingly fast program, on disk in format for S-C Macro Assembler. \$50.

Full Screen Editor. Integrates with the built-in line-oriented editor in the S-C Macro Assembler to provide a powerful full-screen editor for your assembly language source files. Drivers for Videx, STB80, and Apple //e 80-column boards are included, as well as standard 40-column version. Requires 64K RAM in your Apple. Complete source code on disk included. \$50.

S-C Docu-Mentor for Applesoft. Complete documentation of Applesoft internals. Using your ROM Applesoft, produces ready-to-assemble source code with full labels and comments. Educational, entertaining, and extremely helpful. Requires S-C Macro Assembler and two disk drives. \$50.

S-C Word Processor. The one we use for manuals, letters, our monthly newsletter, and whatever. 40-columns only, requires lower-case display and shiftkey mod. Works with standard DOS text files, but at super fast (100 sectors in 7 seconds). No competition to WordStar, but you get complete source code! \$50.

Apple Assembly Line. Monthly newsletter published since October, 1980, for assembly language programmers or those who would like to be. Tutorial articles, advanced techniques, handy utility programs, and commented listings of code in DOS, ProDOS, and the Apple ROMs. Helps you get the most out of your Apple! \$18/year.

S-C SOFTWARE CORPORATION
2331 Gus Thomasson, Suite 125
Dallas, TX 75228 (214) 324-2050

Professional Apple Software Since 1978
Visa, MasterCard, American Express, COD accepted

Apple is a trademark of Apple Computer, Inc.



I do not know why, but the TXS instruction transfers the entire 16-bit value of X to S when you are in the 65802 mode, regardless of the status of the M and X bits. Since M and X are both 1, the high byte of the X-register is 00. Therefore the TXS instruction at \$9FB9 in DOS clears the high byte of the S-register. The RTS at \$9FC4 then uses a return address from page zero, rather than page one.

I tried various experiments to see how TXS and TSX worked, and also examined TXA and TAX. In my humble opinion, the 65802 is inconsistent here. If you are in 65802 mode with M and X = 1, TXA does not modify the high byte of the A-register. This is what I expect and what I want. But TXS does modify the high byte of the S-register, contrary to my expectations.

Of course, as long as you know exactly how the chip works it really doesn't matter a lot. The problems come when we ASSUME we know how it works, but are wrong. The best antidote for these kind of assumptions, at least until a definitive reference manual for the chip is published, is trial and error.

I have had my 65802 for about six months now, and still have had no problems whatsoever with compatibility as long as I stay in normal 6502 mode. If I leave it in 65802 and go charging through a program written for the 6502 mode, I expect I will run into trouble.

8086/8088 Cross Assembler

Use your Apple to learn 8086 programming! You can program for the IBM PC, the clones, and ALF's co-processor board without ever leaving the friendly environment of Apple DOS 3.3.

This easy-to-use cross assembler, based on the S-C Assembler II (Version 4.0), covers all the 8086 and 8088 instructions and all the addressing modes. Instruction mnemonics are based on the Microsoft 8086 assembler. Does not include newer S-C Assembler features like macros or the EDIT command.

Documentation covers the differences from standard S-C Assembler operation and syntax. Sample source programs help you become familiar with the assembler syntax.

With permission from S-C Software, XSM 8086/8088 is available to owners of any S-C Assembler for \$80.00 post-paid. (No credit cards or purchase orders.)

Don Rindsberg
The Bit Stop
5958 S. Shenandoah Rd.
Mobile, AL 36608

(205) 342-1653

Interrupt TraceCharles H. Putney
Dublin, Ireland

Have you ever wondered what's happening when the Apple goes off into nothingness? If your answer is yes, then this short utility will help you find out.

I was recently debugging an assembly language program and ran into this problem. The program seemed to work for almost all the input data, but occasionally would hang. After several frustrating hours trying to simulate the event, I decided that an interrupt trace utility would solve my problems. Later when I had this utility working, it was easy to see why the program was hanging.

This utility consists of a pushbutton addition to the Apple which connects to the interrupt request line (IRQ) of the 6502 and an interrupt service routine which is in page three. When the interrupt pushbutton is depressed, the interrupt service routine displays the program counter and all the registers on the bottom line of the screen. It also displays a flashing cursor and waits for an "S" or "G" from the keyboard to stop or resume execution.

I have mounted a pushbutton switch at the upper right hand side of the keyboard in the center of the styling surface. For a temporary installation I suggest leaving the pushbutton on a flexible lead. The wiring is easily done with 30 gage wire wrap wire. One side of the pushbutton is connected to ground. You may solder a wire to any convenient ground point on the top of the circuit board. Or, for a temporary installation, you could stick a wire into pin 8 of the game I/O connector.

The other side of the pushbutton is connected to the IRQ signal. I found that signal at pin 4 of the 6502. Remove the 6502 from the socket and strip the insulation from the end of the 30 gage wire. Insert it in the socket for the 6502 in pin 4 and replace the 6502 to retain the wire. Route the wire along the chips for a neat installation.

For a temporary hookup, Bob S-C suggests folding a 3-by-5 card in half, and trimming it so that the folded edge just fits into an empty slot. Then, while power to the Apple is off, slip one wire into the space between the card and pin 26 (ground) and the other wire between the card and pin 30 (IRQ). Both of these wires will be on the power-supply side of the card: pin 26 is at the back edge, and pin 30 is the fifth from the back. Once the wires are inserted, you may wish to tape them down.

Enter the routine at address \$300 and BSAVE it. When you want to debug a hanging program, first BRUN the INTERRUPT TRACE utility. This installs the utility at address \$300 to \$3CA. Pressing the pushbutton will cause an immediate display of the current program counter and registers. The utility will wait with a blinking cursor for a "G" or "S" from the keyboard to continue or enter monitor.

Sometimes the program you're investigating may not respond to

the pushbutton. This is because somewhere in the program interrupts have been disabled with the SEI command (\$78). You must search through the entire program and replace these with a CLI instruction (\$58). Make sure that each \$78 found is not data in the program and is a valid instruction before you replace it.

The next time that you have a problem with your Apple "hanging" for no apparent reason, use this utility to see where the 6502 "is". It may help solve those "hard to debug programs".

When you run the program, the SETUP routine (from \$300 to \$30B) sets the interrupt vector location and then enables interrupts. When the pushbutton is depressed, the IRQ line (pin 4 on the 6502) is pulled low. At the completion of the current instruction, the program counter high, program counter low, and processor status are pushed on the stack. Interrupt disable is automatically set and the program counter is loaded with the contents of \$FFFE and \$FFFF. In the Autostart monitor ROM the program counter is set to \$FA40 where the monitor interrupt service routine is located. (In the old monitor the identical routine is at \$FA86) This routine saves the accumulator in \$45 and examines the processor status register to see if the interrupt was caused by a BRK command. Remember, the BRK command shares the same vector location with the interrupt for software simulation of interrupts. If the interrupt was not caused by BREAK then a JMP indirect to location (\$3FE) is performed.

Lines 1280-1290 save the X- and Y-registers. The accumulator has already been saved by the monitor interrupt routine.

Lines 1300-1350 copy the register display titles to the bottom of the screen. Of course, if your program happened to be in one of the full-screen graphics modes, this line will not be visible. If you have a //e, you can add code to sense the graphics mode, save it, switch to text mode; then you will have to restore it all when you type "G" to continue after the interrupt. The new enhanced //e ROMs automatically handle saving and restoring all the bank switched memory, but they still leave the graphics modes up to the programmer.

Lines 1360-1510 convert the values of the five registers and store them into the bottom line. I add 3 to the S-register value before displaying it, so you see the value before the IRQ code pushed PC and S onto the stack. I start with the Y-register pointing at the point on the bottom line where the A-register should be displayed. The DISPLAY.HEX subroutine advances the Y-register by 5, so it is always ready for displaying the next register

Lines 1520-1590 display the PC-register. This value is taken from the stack, where the IRQ automatically saved it.

Lines 1600-1750 wait for you to type "G" or "S". While waiting, the last character on the bottom line is flashed to remind you to type. If you type "G", lines 1760-1800 restore the registers and return to the interrupted program. If you type "S", line 1820 takes you to the monitor.

BETTER THAN RAMWORKS™

MultiRam™ (formerly MaxiRam) offers everything Ramworks does and adds much more with unlimited future use & less power drain. **AVAILABLE NOW** for both the **Ile** and **Ilc** at lower cost from Coit Valley Computers:

	<u>Ramworks</u>	<u>MultiRam Ile</u>
100% Extended 80 Compatible	Yes	Yes
Warranty	3 yr	5 yr
Main Board Maximum	512k	768k
Piggyback Board Maximum	512k	768k + FREE RGB
Total Memory Possible	1024k	1536k (50% MORE!)
AppleWorks™ Expander Software	Up to 735k	UP TO 1100k
Ramdrive Software	\$29	FREE
RGB option	\$129	FREE on Piggyback
65816/8086/6800 CPU Port	No!	Built in FREE
	<u>Ramworks</u> <u>Comparable</u> <u>(Ramdrive Extra)</u>	<u>Coit Valley</u> <u>Computer Price</u> <u>(Ramdrive FREE)</u>
64k MultiRam Ile Card	179.	159.
128k MultiRam Ile Card	249.	215.
256k MultiRam Ile Card	299.	269.
512k MultiRam Ile Card	399.	364.
768k MultiRam Ile Card	653.	458. (WOW!)
1024k MultiRam Ile Card	649. (No RGB)	709. (FREE RGB)
1280k MultiRam Ile Card	No!	797. (FREE RGB)
1536k MultiRam Ile Card	No!	885. (FREE RGB)
256k MultiRam Ilc Card	449. (Z-80)	325. (No Z-80)
512k MultiRam Ilc Card	649. (Z-80)	459. (No Z-80)

MORE MEMORY

Our **FAST** (150 NS) RAM chips, **GUARANTEED** 1 year, can expand MultiRam cards up to 1536k or Ramworks & Legend™ cards up to 1024k! We can expand Multicore™, Z-ram & others also. Easy installation instructions included. **CALL FOR LATEST PRICING.**

	<u>List</u>	<u>Our Price</u>
64k Memory Expander Chips	70.	31.
128k Memory Expander Chips	140.	58.
256k Memory Expander Chips	150.	95.
512k Memory Expander Chips	300.	180.
1024k Memory Expander Chips	600.	300.
AppleWorks™ Program	250.	199.
Ile Enhancement Kit	70.	62.
Pico™ Slim Drive Ile, Ilc	270.	178.
Bernoulli™ Box Macintosh™	1995.	1765.
MultiView™ 80/160 Card	350.	289.
Imagewriter™ (15")	749.	626.
PC Megastore™ 20 Meg Hard Disk		
& 25 Meg Tape Backup II+, Ile	3595.	2889. (WOW!)

Terms: For fastest delivery send Cashier's/Certified check, Money Order. C.O.D. (add \$2) & personal checks accepted (allow 14 days). Add \$3 shipping & phone # to all orders. Add 3% for MasterCard/Visa (include #/expir). Tex res add 6 1/2% tax. **CALL FOR LATEST PRICES!**

MultiRam/Multiview, Ramworks/Z-ram, Appleworks/Macintosh, Legend, Pico, Bernoulli, PC Megastore, Multicore re-spective trademarks of Checkmate Tech, Applied Engineering, Apple Comp, Legend Ind, WGE Int, Iomega, Ampex, Quadram.

COIT VALLEY COMPUTERS
14055 Waterfall Way

(214) 234-5047
Dallas, Texas 75240

We offer a money back guarantee. Try a MultiRAM for 10 days. Return it for re-fund (less \$5.) if not completely satisfied!!

MultiRAM cards also come with a **WRITTEN 5 YR UNCONDITIONAL** guarantee from Checkmate Tech.

They don't care where you got the card from or even if you did happen to save a little money in the process!!!!

For a limited time only, we'll also give an extra 64k of RAM memory with each 256k or 512k MultiRAM card you buy! That's a deal that's hard to beat!!!

```

1000 *SAVE S_IRQ TRAPPER
1010 *-----
1020 *      INTERRUPT TRACE UTILITY
1030 *
1040 *      BY: CHARLES H PUTNEY
1050 *      18 QUINNS ROAD
1060 *      SHANKILL
1070 *      COUNTY DUBLIN
1080 *      IRELAND
1090 *-----
45- 1100 A.REG      .EQ $45      A-REG SAVE AREA USED BY MONITOR
0100- 1110 STACK      .EQ $100     STACK PAGE
03FE- 1120 INTVEC      .EQ $3FE     INTERRUPT VECTOR
07D0- 1130 BOTTOM.LINE .EQ $7D0     LINE 24 OF TEXT SCREEN
1140 *-----
C000- 1150 KEYBD      .EQ $C000     KEYBOARD DATA
C010- 1160 KEYSTB      .EQ $C010     KEYBOARD STROBE
FF69- 1170 MNTR       .EQ $FF69     MONITOR ENTRY POINT (CALL -151)
1180 *-----
1190          .OR $300      PAGE THREE
1200 *-----
0300- A9 0C 1210 SETUP  LDA #INT      LOAD IRQ VECTOR
0302- 8D FE 03 1220      STA INTVEC    LOW BYTE
0305- A9 03 1230      LDA /INT
0307- 8D FF 03 1240      STA INTVEC+1  HIGH BYTE
030A- 58 1250      CLI          ALLOW IRQ'S
030B- 60 1260      RTS
1270 *-----
030C- 8E C5 03 1280 INT     STX XREG      SAVE X (A-REG SAVED BY MONITOR)
030F- 8C C6 03 1290      STY YREG      SAVE Y
1300 *-----DISPLAY REG TITLES-----
0312- A2 27 1310      LDY #39      PUT UP MESSAGE LINE
0314- BD 9D 03 1320      LDA TITLES,X  GET MESSAGE CHAR
0317- 9D D0 07 1330      STA BOTTOM.LINE,X PUT ON SCREEN
031A- CA 1340      DEX
031B- 10 F7 1350      BPL .1      DONE ?
1360 *-----DISPLAY REG VALUES-----
031D- A0 0A 1370      LDY #10      START OF REG DISPLAY AREA
031F- A5 45 1380      LDA A.REG      ...A-REG
0321- 20 7B 03 1390      JSR DISPLAY.HEX
0324- AD C5 03 1400      LDA XREG      ...X-REG
0327- 20 7B 03 1410      JSR DISPLAY.HEX
032A- AD C6 03 1420      LDA YREG      ...Y-REG
032D- 20 7B 03 1430      JSR DISPLAY.HEX
0330- BA 1440      TSX          GET STACK POINTER
0331- E8 1450      INX          POINT AT PROCESSOR STATUS
0332- BD 00 01 1460      LDA STACK,X  ...P-REG
0335- 20 7B 03 1470      JSR DISPLAY.HEX
0338- E8 1480      INX          ADJUST S-REG
0339- E8 1490      INX
033A- 8A 1500      TXA          ...S-REG AS WAS BEFORE INTERRUPT
033B- 20 7B 03 1510      JSR DISPLAY.HEX
1520 *-----DISPLAY PC-REG-----
033E- A0 00 1530      LDY #0      START OF PC-REG DISPLAY
0340- BD 00 01 1540      LDA STACK,X  GET PC HIBYTE
0343- 20 7B 03 1550      JSR DISPLAY.HEX
0346- CA 1560      DEX
0347- A0 02 1570      LDY #2
0349- BD 00 01 1580      LDA STACK,X  GET PC LOBYTE
034C- 20 7B 03 1590      JSR DISPLAY.HEX
1600 *-----WAIT FOR "S" OR "G"-----
034F- AD 00 C0 1610      LDA KEYBD    KEY PRESSED ?
0352- 10 0B 1620      BPL .3      NO
0354- 8D 10 C0 1630      STA KEYSTB  CLEAR THE KEY
0357- C9 C7 1640      CMP #*G"      GO AHEAD ?
0359- F0 14 1650      BEQ .4      YES
035B- C9 D3 1660      CMP #*S"      STOP ?
035D- F0 19 1670      BEQ .5      YES
035F- CA 1680      DEX          BLINK CURSOR
0360- D0 FD 1690      BNE .3
0362- 88 1700      DEY          LONGER DELAY
0363- D0 FA 1710      BNE .3
0365- AD F7 07 1720      LDA BOTTOM.LINE+39 LAST CHAR ON SCREEN
0368- 49 80 1730      EOR #$80      INVERT IT
036A- 8D F7 07 1740      STA BOTTOM.LINE+39 REPLACE IT
036D- D0 E0 1750      BNE .2      BRANCH ALWAYS

```

```

1760 *---"G" TYPED, RETURN-----
036F- A5 45 1770 .4 LDA A.REG RESTORE THE REGISTERS
0371- AE C5 03 1780 LDX XREG
0374- AC C6 03 1790 LDY YREG
0377- 40 1800 RTI BACK TO WORK
1810 *---"S" TYPED, SO STOP-----
0378- 4C 69 FF 1820 .5 JMP MNTR ENTER THE MONITOR
1830 *-----
1840 DISPLAY.HEX
037B- 48 1850 PHA SAVE THE A-REG
037C- 4A 1860 LSR SHIFT INTO LOWER NIBBLE
037D- 4A 1870 LSR
037E- 4A 1880 LSR
037F- 4A 1890 LSR
0380- 20 93 03 1900 JSR DIGIT MAKE IT A DIGIT
0383- 99 D0 07 1910 STA BOTTOM.LINE.Y SHOW HIGH NIBBLE
0386- 68 1920 PLA GET A-REG AGAIN
0387- 29 0F 1930 AND #0F MASK IT
0389- 20 93 03 1940 JSR DIGIT MAKE IT A DIGIT
038C- 99 D0 07 1950 STA BOTTOM.LINE.Y SHOW LOWER NIBBLE
038F- C8 1960 INY
0390- C8 1970 INY
0391- C8 1980 INY
0392- 60 1990 RTS
2000 *-----
0393- C8 2010 DIGIT INY
0394- 09 B0 2020 ORA #$B0 ADD NUMBER ZERO
0396- C9 BA 2030 CMP #$BA IS IT A LETTER
0398- 90 02 2040 BCC .1 NO - DONE
039A- 69 06 2050 ADC #$6 6 PLUS CARRY MAKES A
039C- 60 2060 .1 RTS
2070 *-----
039D- A0 A0 A0
03A0- A0 A0 AD
03A3- A0 A0 A0
03A6- C1 BD A0
03A9- A0 A0 D8
03AC- BD A0 A0
03AF- A0 D9 BD
03B2- A0 A0 A0
03B5- D0 BD A0
03B8- A0 A0 D3
03BB- BD A0 A0
03BE- A0 A0 A0
03C1- A0 A0 A0
03C4- A0 A0 A0
2080 TITLES .AS -/ - A= X= Y= P= S= /
2090 *-----
03C5- 00 2100 XREG .DA #-# X REGISTER SAVE AREA
03C6- 00 2110 YREG .DA #-# Y REGISTER SAVE AREA

```

 WORDPAK - The Wordprocessor Plus

1. Switch freely between 40-col and 80-col screen
2. Write formletters, even personalized formletters
3. Chain files and create extra large documents
4. Create, maintain, and access your own data base
5. Insert printer or typesetting codes into text
6. Formfill mode allows filling fields in forms
7. Print address blocks on letters and envelopes
8. Mailing labels: sorted, also printed selectively
9. Data base allows up to 1.118 addresses per disk
10. Footnotes at bottom of pages or at end of text
11. Large textbuffer holds over 31.000 characters
12. Fast? - Whoa! - Loads itself in just 4 seconds
13. For Apple //e, //c, or Apple][+ w/Language Card

- And much more. See your dealer, or order direct.
 \$199.95 + \$1.50 shipping/handling charge via UPS (US)
 Mid-West Marketing, 1492 Ammons, Denver, CO. 80215

Improving the Single-Byte Converter.....Bruce Love New Zealand

Bob's single byte converter (see Jan 85 issue, pages 31-32) can be shortened by one byte. The left column is from Bob's code. the right a shorter version:

1040 .1	LDX #0"	.1	LDX #0"-1
1050 .2	CMP DECTBL,Y		SEC
1060	BCC .3	.2	SBC DECTBL,Y
1070	SBC DECTBL,Y		INX
1080	INX		BCS .2
1090	BNE .2		ADC DECTBL,Y

I also tried a different approach, using the decimal mode to count tens, then printing the tens as a hex value with the monitor routine at \$FDDA and the remainder (units digit) with \$FDED. This routine takes longer time, but does not need to use the X-register.

00-	1000	BYTE	.EQ \$00
FD8E-	1010	COUT	.EQ \$FDED
FD8E-	1020	CROUT	.EQ \$FD8E
FDDA-	1030	PRBYTE	.EQ \$FDDA
	1040	#-----	
0800- A9 00	1050	P	LDA #0
0802- 85 00	1060		STA BYTE
0804- 20 11 08	1070 .1		JSR PRINT.000.255
0807- 20 8E FD	1080		JSR CROUT
080A- E6 00	1090		INC BYTE
080C- A5 00	1100		LDA BYTE
080E- D0 F4	1110		BNE .1
0810- 60	1120		RTS
	1130	#-----	
	1140		PRINT.000.255
0811- A0 00	1150		LDY #0
0813- 38	1160		SEC
0814- E9 0A	1170 .1		SBC #10
0816- 08	1180		PHP
0817- 48	1190		PHA
0818- 98	1200		TYA
0819- F8	1210		SED
081A- 69 00	1220		ADC #0
081C- A8	1230		TAY
081D- 68	1240		PLA
081E- 28	1250		PLP
081F- B0 F3	1260		BCS .1
0821- 69 BA	1270		ADC #0+10
0823- 48	1280		PHA
0824- 98	1290		TYA
0825- 20 DA FD	1300		JSR PRBYTE
0828- 68	1310		PLA
0829- 4C ED FD	1320		JMP COUT

AppleVisions, a Glimpse

Here is a very elementary introduction to Apple Assembly Language programming by that old master Bob Bishop, along with Linda Grossberger and Harry Vertelney. This 150+ page book and its companion diskette gently and humorously guide the beginning programmer into the realm of machine code. A "Cardboard Computer" introduces the concepts of registers, machine instructions, addressing, and branching. This background is then applied to the Apple's 6502 and the surrounding computer. AppleVisions is a nice place for the absolute beginner to start, especially the younger programmers interested in finding out what assembly language is.

AppleVisions. Addison-Wesley, 1985. \$39.95 including diskette.

Two ROM Sets in One Apple //e.....Bob Sander-Cederlof

If and when you decide to upgrade to the new enhanced //e ROMs (which Apple sells for \$70 along with a 65C02), you will probably have to turn your old ROMs over to the store that makes the switch. Reportedly, Apple is binding the stores with a contract that forces them to collect all the old chips.

That is VERY unfortunate. It could lead to wild shouting and panic, when you discover some of your favorite old software no longer works.

The upgrade consists of three parts:

- * the new processor chip (65C02), which is nice but not especially useful until software which uses its new features becomes available;
- * a new character generator ROM which includes special characters for icons and line drawing in text mode (called the "mouse" characters).
- * new CD and EF ROMs which upgrade the firmware.

The new firmware does NOT use any of the new features in the 65C02, so you could use it without the new cpu chip. Furthermore, there is no absolute requirement to have the new character generator installed. The new firmware is much better than the old, having lost some bugs and speeded up the 80-column scrolling and added lower-case support to Applesoft (among other things). It is compatible with the 6502, the 65C02, and the new 65802.

I personally do not yet have any use for the mouse characters, and do not expect to. Don Lancaster, in the June 1985 issue of "Computer Shopper", tells how to connect a 2764 EPROM in the character generator socket. The 2764 can hold two complete character sets, because it has twice the capacity of the 2732 normally in that socket. However, the socket has only 24 holes and the 2764 has 28 pins! Don shows how to wire this up with a socket adapter, and use a toggle switch to select either half.

And now Apple has "sort of" released an even more enhanced set of firmware, with debugging stuff built in. You may not see them on the open market for some time, but I like them even better than the standard enhanced ROMs. The "debug" ROMs add an absolute RESET (ctrl-RESET with solid apple), 16-byte hex display in the monitor when in 80-column mode, display of both hex and ASCII values of each byte in a memory dump, and the ability to use all monitor commands on both main and auxiliary memory. The disassembler and miniassembler are both present, and enhanced to include the 65C02 extensions.

The CD and EF ROM sockets are compatible with 2764 EPROMs. You can also use 27128s, which have twice the space. Pin 26 on the 2764 is always tied to +5 volts. On a 27128, pin 26 selects the top or bottom half of the 16K bytes inside. You can burn one set of firmware in one half, and the other set in the other

half. Then bend out pin 26 a little, so that it does not go into the socket when you insert the chip. Attach a clip lead to the bent-out pin, and connect the other end to either +5 volts or ground, to select the half you want at any given time.

You can connect it to a toggle switch, or just stick the bare end of a wire into the game paddle connector. If you use the game socket on the motherboard, pin 1 is +5 volts and pin 8 is ground. Or stick a wire into one of the annunciator outputs (pins 12, 13, 14, and 15) so you can flip back and forth between firmware sets by software control.

It can be a little tricky to make a copy of the ROM firmware and get it into RAM or on a disk, so that you can later burn it in your own EPROM. Especially in the Cxxx part. My approach, since I have more than one Apple, is to put my SCRG PromGramer card in a different machine. Then one by one I can read the //e ROMs and burn them into the appropriate 27128s. This a lot faster than trying to figure out how to flip all the //e soft switches so as to get at the different banks of Cx ROM code.

I have recently seen 27128s priced as low as \$5 and as high as \$20, in the back of Byte magazine. It is well worth it to invest in a PromGramer, at \$140, and an EPROM eraser (\$50 to \$100 from Logical Devices in Florida, see Byte ads). You can keep your Apple standard for commercial software, and still have your own private firmware on the motherboard at the flip of a switch!

Just after finishing this article I happened to be looking up something else in Understanding the Apple //e, by Jim Sather. There, big as life, on page 6-11, is a description of the same method for installing modified ROMs.

Assembly Corner

Technical Products
for the Apple and Macintosh Computers

11601 N.W. 18th St.
Pembroke Pines, FL 33026

Assembly Corner is now offering two courses on Assembly language for the Apple II series of computers.

1. Assembly for the serious beginner -
a one year course on twelve monthly disks, each one packed with lessons, information, source code, and actual programs as examples. Emphasis is on learning practical techniques that you can USE.
2. Graphics and Animation Series -
A four month course on four disks, filled with information on Apple graphics techniques, ROM routines, HPLOT animation, raster shapes, pre-shifted shapes and more. Includes special programmer's utility not available elsewhere. This course requires an intermediate understanding of Assembly Language.

A FREE ASSEMBLY CORNER POSTER WITH EACH ORDER!

Both courses require an Apple II (any kind), a printer, and one disk drive. A color monitor or TV is recommended for the Graphics Series. For more details call (305) 431-6892

- | | |
|--|----------------|
| <input type="checkbox"/> One month trial - Beginners' series | ----- \$15.00 |
| <input type="checkbox"/> Full year - Beginners' series | ----- \$180.00 |
| <input type="checkbox"/> One month trial - Graphics series | ----- \$6.00 |
| <input type="checkbox"/> Four months - Graphics series | ----- \$50.00 |

Windows



For the Apple II computer

Windows is a machine language utility program for Basic programmers. It adds ten new commands to Applesoft Basic, which allow you to create windows in the Apple test screen. Display any message, menu, etc. right over the existing text, without destroying the original display. Text inside windows scrolls independently. In addition, Windows allows you to split the screen vertically and horizontally, and use each half separately, and you can create other special effects too! The possibilities are endless. If you program in Basic, you'll love doing Windows!

Yes! I want to do Windows on my Apple! Here's my check for \$19.95.

Assembly Corner
11601 N.W. 18th St.
Pembroke Pines, FL 33026
(305) 431-6892

**A CALL Utility for Applesoft.....David C. Johnson
Applied Engineering**

Anyone who has ever used Applesoft eventually realizes that the most powerful statement in the language is CALL. It allows you to get to the Monitor for instance (however, Extended Debugging Monitor users have a better way). When writing a program in BASIC, you invariably will want to do something that is at best difficult and often impossible to code using the other Applesoft statements.

The solution to this type of situation is to speak to your Apple in its native tongue. There are several ways this can be done. Ampersand (&) routines are a popular technique. The USR(function even has its uses. The most logical way, for me, is the CALL statement.

Using CALL neatly transfers control from the Applesoft interpreter to whatever you want to do in machine language. The one disadvantage to CALL is that the processor's registers do not contain useful data when your machine code gets control.

The CALL utility presented in this article will allow you to specify, as part of the CALL statement, the contents of any or all of the registers upon entry of your machine language subroutine. You assign the register contents with LET-like structures. Obviously you can only fit an 8 bit value into the 8 bit registers and the program counter value will probably be a 16 bit number. Here's how the CALL statement should be written:

CALL 768,PC=word,A=byte,X=byte,Y=byte,P=byte

The expressions "word" and "byte" may be any valid Applesoft numeric expression. This is because the utility calls routines internal to Applesoft to evaluate the expressions. If an expression results in a value larger than the register to which it is being assigned, or isn't numeric, or is invalid, you will get one of the usual errors. The commas shown separating the register assignments are required (syntax error if comma missing). The equals characters ("=") are also required. The register names (PC, A, X, Y, & P) must be upper case on older Apples, while the newer firmware will convert lower case for you (or in spite of you). The register assignments may appear, after the first comma, in any order and need not all be specified. Unspecified registers will be loaded with their last used value. Previously unused registers default to zero, except the P-register which defaults to \$04 in order to set the interrupt disable flag.

The program is well commented, but I'll add one more note of caution. Readers with Apples containing regular 6502s (not 65C02s or 65802s) should avoid re-assembling the code with the label PC.Sav's bytes falling across a page boundary (\$XXFF).

The program was written using the ProDOS version of the S-C Macro Assembler 2.0, while I was beta testing it for Bob. It works GREAT!

NEW DON LANCASTER RELEASES

Available **ONLY** from Synergetics. All software open and unlocked.

ABSOLUTE RESET (IIC/old IIE/new IIE) \$19.50

Get back in total control. No more hole blasting! Access any part of any program at any time for any reason. Passes all diagnostics. Invisible till activated. Includes EPROM burner adapter details, service listings, and a free bonus book.

APPLEWRITER™ IIE TOOLKIT \$39.50

EIGHT diskette sides include a complete disassembly script, self-prompting glossaries, Diablo microjustify and proportional space, patches for NULL, shortline, IIC detrashing, answers to lots of help line questions, source code capturing, two columns, WPL secrets, space on disk, keyword indexing, more.

ProDOS APPLEWRITER™ 2.0 TOOLKIT \$39.50

Another EIGHT diskette sides crammed full. Includes a neat automatic source code capturer, a complete disassembly script, dozens of patches, prefix hassle fixes, Grappler and shortline repairs, NULL diversion, two columns, auto indexing, plus Don's unique self-prompting glossaries, Diablo microjustify, etc.

Both TOOLKITS \$59.50

APPLEWORKS™ DISASSEMBLY SCRIPT \$49.50

TEN diskette sides chock full of Applework's innermost secrets, using Don's unique and powerful "tearing method". Primarily for gonzo hackers and definitely NOT for the faint of heart.

COMPANION DISKETTES:

For Enhancing Your Apple II, Volume I \$19.50

For Enhancing Your Apple II, Volume II \$19.50

For Don Lancaster's Assembly Cookbook \$19.50

ASSORTED GOODIES:

VAPORLOCK instant sync package \$19.50

Old Fangled Animation demo \$ 9.50

The Incredible Secret Money Machine (autographed) \$ 7.50

Laserwriter/Applewriter Utilities Package \$24.50

SYNERGETICS
746 First Street
Box 809-AAL
Thatcher, AZ, 85552

FREE
VOICE HELPLINE
(602) 428-4073

VISA and MASTERCHARGE accepted. Please - no COD, foreign, or purchase orders.
Appleworks, Applewriter, and ProDOS are registered trademarks of Apple Computer.

```

1000 *SAVE S.CALL.UTIL
1010
1020 * 6/13/85 dcj
1030
1040 * CALL 768{,pc=word,a=byte,x=byte,y=byte,p=byte}
1050
1060 -----
1070
1080 .OR $300
1090 .TF CU
1100
D0- 1110 EQ.TOK .EQ $D0 Applesoft '=' token
1120
B1- 1130 CHRGET .EQ $B1 -$C8 advance TTXPTR & fetch chr
B7- 1140 CHRGOT .EQ $B7 just fetch chr
1150
DD67- 1160 FRMNUM .EQ $DD67 evaluate FP expression (FAC)
DEC0- 1170 SYNCHR .EQ $DEC0 require chr in Acc syntax @ TTXPTR
DEC9- 1180 SYNERR .EQ $DEC9 syntax error
E6F8- 1190 GETBYT .EQ $E6F8 evaluate 8 bits @ TTXPTR (X-reg)
E752- 1200 GETADR .EQ $E752 convert FAC to 16 bits in Acc & Y-reg
1210
1220 -----
1230
1240 CALL.UTIL
1250
0300- 20 B7 00 1260 JSR CHRGOT get chr after call adr expression
0303- C9 2C 1270 CMP #'.' comma indicates more stuff follows
0305- F0 11 1280 BEQ .1 =>go continue parsing
0307- AD 7D 03 1290 LDA P.SAV load registers
030A- 48 1300 PHA (P-reg via stack)
030B- AD 7A 03 1310 LDA ACC.SAV
030E- AE 7B 03 1320 LDX X.SAV
0311- AC 7C 03 1330 LDY Y.SAV
0314- 28 1340 PLP
0315- 6C 7E 03 1350 JMP (PC.SAV) go 4 it!
1360
* we got something to parse
1370
0318- 20 B1 00 1390 .1 JSR CHRGET get chr after comma
031B- C9 41 1400 CMP #'A' (as in 'Acc')
031D- F0 0F 1410 BEQ .2 =>go get '=' & byte for Acc
031F- C9 58 1420 CMP #'X' (as in 'X-reg')
0321- F0 13 1430 BEQ .3 =>go get '=' & byte for X-reg
0323- C9 59 1440 CMP #'Y' (as in 'Y-reg')
0325- F0 17 1450 BEQ .4 =>go get '=' & byte for Y-reg
0327- C9 50 1460 CMP #'P' (as in P-reg or Program Counter)
0329- F0 1B 1470 BEQ .5 =>go get '=' or 'C'...
032B- 4C C9 DE 1480 JMP SYNERR razz
1490
* pickup Acc byte
1500
032E- 20 6A 03 1520 .2 JSR .7 require '=' (@ next) & fetch byte exp
0331- 8E 7A 03 1530 STX ACC.SAV stuff it
0334- 50 CA 1540 BVC CALL.UTIL ...always
1550
* pickup X-reg byte
1560
0336- 20 6A 03 1580 .3 JSR .7 require '=' (@ next) & fetch byte exp
0339- 8E 7B 03 1590 STX X.SAV stuff it
033C- 50 C2 1600 BVC CALL.UTIL ...always
1610
* pickup Y-reg byte
1620
033E- 20 6A 03 1640 .4 JSR .7 require '=' (@ next) & fetch byte exp
0341- 8E 7C 03 1650 STX Y.SAV stuff it
0344- 50 BA 1660 BVC CALL.UTIL ...always
1670
* Finish parsing 'P=' or 'PC='
1680
0346- 20 B1 00 1700 .5 JSR CHRGOT advance to next chr position & fetch it
0349- C9 43 1710 CMP #'C' (as in 'Program Counter')
034B- F0 0B 1720 BEQ .6 =>go get '=' & 16 bits for PC
1730
* pickup P-reg byte
1740
034D- 20 75 03 1760 JSR .10 require '=' @ current chr position
0350- 20 6D 03 1770 JSR .8 fetch byte expression
0353- 8E 7D 03 1780 STX P.SAV stuff it
0356- 50 A8 1790 BVC CALL.UTIL ...always

```

```

1800
1810 * pickup PC word
1820
0358- 20 72 03 1830 .6 JSR .9 require '=' @ next chr position
035B- 20 67 DD 1840 JSR FRMNUM fetch FP expression
035E- 20 52 E7 1850 JSR GETADR convert FP expression to Acc & Y-reg
0361- 8C 7E 03 1860 STY PC.SAV stuff 'em
0364- 8D 7F 03 1870 STA PC.SAV+1
0367- 4C 00 03 1880 JMP CALL.UTIL no flag known...
1890
036A- 20 72 03 1900 .7 JSR .9 require '=' @ next chr position
1910
036D- 20 F8 E6 1920 .8 JSR GETBYT fetch byte expression (2 X-reg)
0370- B8 1930 CLV to force branch
0371- 60 1940 RTS
1950
0372- 20 B1 00 1960 .9 JSR CHRGET 1st advance to next chr position
1970
0375- A9 D0 1980 .10 LDA #EQ.TOK require '=' before register expressions
0377- 4C C0 DE 1990 JMP SYNCHR (SYNTAX ERROR IF '=' NOT FOUND)
2000
2010 #-----
2020
037A- 00 2030 ACC.SAV .DA #00
037B- 00 2040 X.SAV .DA #00
037C- 00 2050 Y.SAV .DA #00
037D- 04 2060 P.SAV .DA #04
037E- 00 00 2070 PC.SAV .DA #0000
2080
2090 #-----

```

WARNING!

Some of Applied Engineering's Products are
being sold by non-authorized dealers.

To insure your warranty, to receive upgrades and
to obtain technical support, please buy direct from
Applied Engineering or an authorized dealer.

Be suspicious if:

- 1) The dealer is mail order.
- 2) You don't know the dealer.
- 3) The product is not in stock.

If you're not sure about a dealer, please call us.

(214) 241-6060



APPLIED ENGINEERING
"We Set the Standard"

Some Final DP18 Subroutines.....Bob Sander-Cederlof

Gerald Ferrier (Princeton, Minnesota) wrote to point out that we somehow omitted a double-handful of subroutines from our lengthy series on 18-digit arithmetic for Applesoft. With apologies to you all, and thanks to Gerald, here they are:

```

1000 *SAVE DP18.MOVE.SUBS
2020 *-----
2030 *      MOVE (Y,A) INTO DAC. YA IS UNPACKED
2040 *-----
2050 MOVE.YA.DAC.1
2060     STA PNTR
2070     STY PNTR+1
2080     LDY #10      MOVE 11 BYTES
2090 .1    LDA (PNTR),Y
2100     STA DAC,Y
2110     DEY
2120     BPL .1
2130     LDA DAC.EXPONENT
2140     STA DAC.SIGN
2150     AND #$7F
2160     STA DAC.EXPONENT
2170     RTS
2180 *-----
2190 *      MOVE (Y,A) INTO ARG. YA IS UNPACKED
2200 *-----
2210 MOVE.YA.ARG.1
2220     STA PNTR
2230     STY PNTR+1
2240     LDY #10      MOVE 11 BYTES
2250 .1    LDA (PNTR),Y
2260     STA ARG,Y
2270     DEY
2280     BPL .1
2290     LDA ARG.EXPONENT
2300     STA ARG.SIGN
2310     AND #$7F
2320     STA ARG.EXPONENT
2330     RTS
2340 *-----
2350 *      MOVE DAC TO (Y,A) WITHOUT PACKING
2360 *-----
2370 MOVE.DAC.YA.1
2380     STA PNTR
2390     STY PNTR+1
2400     LDA DAC.EXPONENT
2410     BPL .0
2420     JMP DAC.YA.O.U OVER- OR UNDER-FLOW
2430 .0    BIT DAC.SIGN
2440     BPL .1      POSITIVE
2450     ORA #$80    NEGATIVE
2460 .1    LDY #0
2470 .2    STA (PNTR),Y
2480     INY
2490     LDA DAC.Y
2500     CPY #11     11 BYTES
2510     BCC .2
2520     RTS
2530 *-----
2540 MOVE.DAC.TEMP1
2550     LDA #DP.TEMP1
2560     LDY /DP.TEMP1
2570     JMP MOVE.DAC.YA.1
2580 *-----
2590 MOVE.TEMP1.ARG
2600     LDA #DP.TEMP1
2610     LDY /DP.TEMP1
2620     JMP MOVE.YA.ARG.1
2630 *-----
2640 MOVE.TEMP1.DAC
2650     LDA #DP.TEMP1
2660     LDY /DP.TEMP1
2670     JMP MOVE.YA.DAC.1
2680 *-----
2690 MOVE.DAC.TEMP2
2700     LDA #DP.TEMP2
2710     LDY /DP.TEMP2
2720     JMP MOVE.DAC.YA.1
2730 *-----
2740 MOVE.TEMP2.DAC
2750     LDA #DP.TEMP2
2760     LDY /DP.TEMP2
2770     JMP MOVE.YA.DAC.1
2780 *-----
2790 MOVE.TEMP2.ARG
2800     LDA #DP.TEMP2
2810     LDY /DP.TEMP2
2820     JMP MOVE.YA.ARG.1
2830 *-----
2840 MOVE.TEMP3.DAC
2850     LDA #DP.TEMP3
2860     LDY /DP.TEMP3
2870     JMP MOVE.YA.DAC.1
2880 *-----
2890 MOVE.TEMP3.ARG
2900     LDA #DP.TEMP3
2910     LDY /DP.TEMP3
2920     JMP MOVE.YA.ARG.1
2930 *-----
2940 MOVE.DAC.TEMP3
2950     LDA #DP.TEMP3
2960     LDY /DP.TEMP3
2970     JMP MOVE.DAC.YA.1
2980 *-----

```

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is \$18 per year in the USA, sent Bulk Mail; add \$3 for First Class postage in USA, Canada, and Mexico; add \$14 postage for other countries. Back issues are available for \$1.80 each (other countries add \$1 per back issue for postage).

All material herein is copyrighted by S-C SOFTWARE CORPORATION. all rights reserved. (Apple is a registered trademark of Apple Computer, Inc.)